ARMY RESEARCH LABORATORY

The Inertial Reticle Technology (IRT) Applied to an M16A2 Rifle Firing From a Fast Attack Vehicle

by Timothy L. Brosseau, Mark D. Kregel,
Baily T. Haug, and John T. McLaughlin

ARL-TR-2209 April 2000

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Abstract

Motion of the muzzle of a weapon fired from a moving vehicle occurs during firing because of many factors, such as vibrations caused by the vehicle's wheels or the terrain. This motion can have adverse effects on the capabilities of the weapon to hit a target, because the shooter is unable to accurately position the muzzle of the weapon onto the target as the projectile exits the barrel. Large, heavy vehicles, such as the Abrams tank, the Bradley Fighting Vehicle, and the costly Apache helicopter, have very expensive gun turrets that are controlled by very expensive, fully stabilized gun sights to accurately position the muzzle of the weapon onto the target. However, small and lightweight vehicles, such as a small helicopter, a fast attack vehicle, or a high-mobility multipurpose wheeled vehicle (HMMWV), cannot justify such expensive gun turrets and fully stabilized sights. Therefore, to improve the accuracy of a weapon firing from a small, lightweight vehicle, the U.S. Army Research Laboratory (ARL) has developed the Inertial Reticle Technology (IRT).

This report presents how the IRT was applied to a 5.56-mm M16A2 rifle firing from a fast attack vehicle. The complete details of the IRT applied to a 5.56-mm M16A2 rifle firing from a fast attack vehicle are presented along with an analysis of stationary and moving vehicle live fire test data.

Table of Contents

INTENTIONALLY LEFT BLANK.

List of Figures

INTENTIONALLY LEFT BLANK.

1. Introduction

Motion of the muzzle of a weapon fired from a moving vehicle occurs during firing because of many factors, such as vibrations caused by the vehicle's wheels or the terrain. This motion can have adverse effects on the capabilities of the weapon to hit a target, because the shooter is unable to accurately position the muzzle of the weapon onto the target as the projectile exits the barrel. Large, heavy vehicles, such as the Abrams tank, the Bradley Fighting Vehicle, and the costly Apache helicopter, have very expensive gun turrets that are controlled by very expensive, fully stabilized gun sights to accurately position the muzzle of the weapon onto the target. However, small and lightweight vehicles, such as a small helicopter, a fast attack vehicle, or a high-mobility multipurpose wheeled vehicle (HMMWV), cannot justify such expensive gun turrets and fully stabilized sights. Therefore, to improve the accuracy of a weapon firing from a small, lightweight vehicle, the U.S. Army Research Laboratory (ARL) has developed the Inertial Reticle Technology (IRT).

2. The IRT Applied to a Weapon Firing From a Lightweight Vehicle

Two test beds were built as part of the IRT program, which was an Army Science and Technology Objective culminating in December 1998. The initial test bed was a fast attack vehicle on which the IRT was integrated with a 5.56-mm M16A2 rifle. The second test bed was a HMMWV on which the IRT was integrated with a .50-cal. M2 heavy-barrel machine gun. This section discusses the common aspects of the design, and the rest of this report focuses on the fast attack vehicle test bed.

The IRT replaces the conventional sights or scope on the weapon fired from a lightweight vehicle with a video camera that is mounted to the weapon and a video display mounted in a remote control panel that rests on the gunner's lap inside the vehicle. The IRT also replaces the weapon mount on the lightweight vehicle with a remotely operated, lightweight, and inexpensive weapon positioner or turret.

In the IRT test beds, the weapon positioner drives the weapon in both the elevation and azimuth directions by means of two inexpensive low-power stepper motors. Two shaft angle encoders are attached to the elevation and azimuth axes of the weapon positioner to measure the relative angular displacements of the weapon relative to the weapon platform. Three orthogonal angular rate sensors are mounted on the weapon platform. The outputs of the rate sensors are integrated to provide the angular displacements of the weapon platform in the pitch, yaw, and roll directions. On the fast attack vehicle, a simple wheel counter, mounted on the vehicle, counts wheel rotations, which allows the calculation of vehicle translation relative to the target. A revolution counter on the speedometer cable performs the same task on the HMMWV.

The initial range of the vehicle to the target is put into the computer manually or from a range finder mounted on the weapon. A continuous readout of the range is shown on the operator's display in the remote control panel that rests on the gunner's lap inside the vehicle. A small computer is used to generate two electronic pointers that can be overlaid on the video image. The first pointer is a dot that is aligned with the barrel centerline of the weapon and, thus, represents the aim point. With inputs from the range finder, the wheel counter, and the shaft angle encoders, the aim point is ballistically corrected for range and lag angles and becomes the ballistic solution. The second pointer is a crosshair referred to as the inertial reticle. This reticle is driven in opposition to the weapon and vehicle motions, as measured by the shaft angle encoders, integrated rate sensors, and the wheel counter, so that the inertial reticle appears to remain fixed relative to the target, even though the weapon and vehicle might be moving. Even though the video scene may be moving around significantly, there is no relative motion between the inertial reticle and the target. This makes it easy to position the inertial reticle over the desired target using a joystick mounted on the remote control panel.

Once the initial range to the target is put into the computer and the inertial reticle is accurately placed over the desired target, the computer continuously measures the difference in the position of the inertial reticle relative to the aim point. This error signal is used to drive the stepper motors to position the aim point over the inertial reticle. As the aim point and the inertial reticle are aligned, a prediction algorithm on the computer calculates the precise firing time,

ensuring that the projectile exits the muzzle as the ballistic solution is aligned with the intertial reticle. Assuming that the gunner has enabled the system to fire, the precise firing is accomplished by means of an electric solenoid that is attached to the trigger of the weapon. To simplify the display for the gunner, the ballistic solution is typically not displayed, and the gunner's tasks are to select the target, enter the range, and keep the inertial reticle on target.

3. The IRT Applied to a 5.56-mm M16A2 Rifle Firing From a Fast Attack Vehicle

The IRT was applied to a 5.56-mm M16A2 rifle firing from a fast attack vehicle, as shown in Figures 1 and 2. The 5.56-mm M16A2 rifle was fitted with a Sony (EVI-330T) CCD camera block. The Sony (EVI-330T) camera block has a 12× optical zoom, auto focus lens. It also has a 2× electronic zoom, which when combined with the 12× optical zoom gives a video image that is equivalent to the image seen through a conventional 12× scope. The video image from the camera block is viewed by the gunner on a Sony flat-panel display monitor (FMD-402A) mounted in the remote control panel.

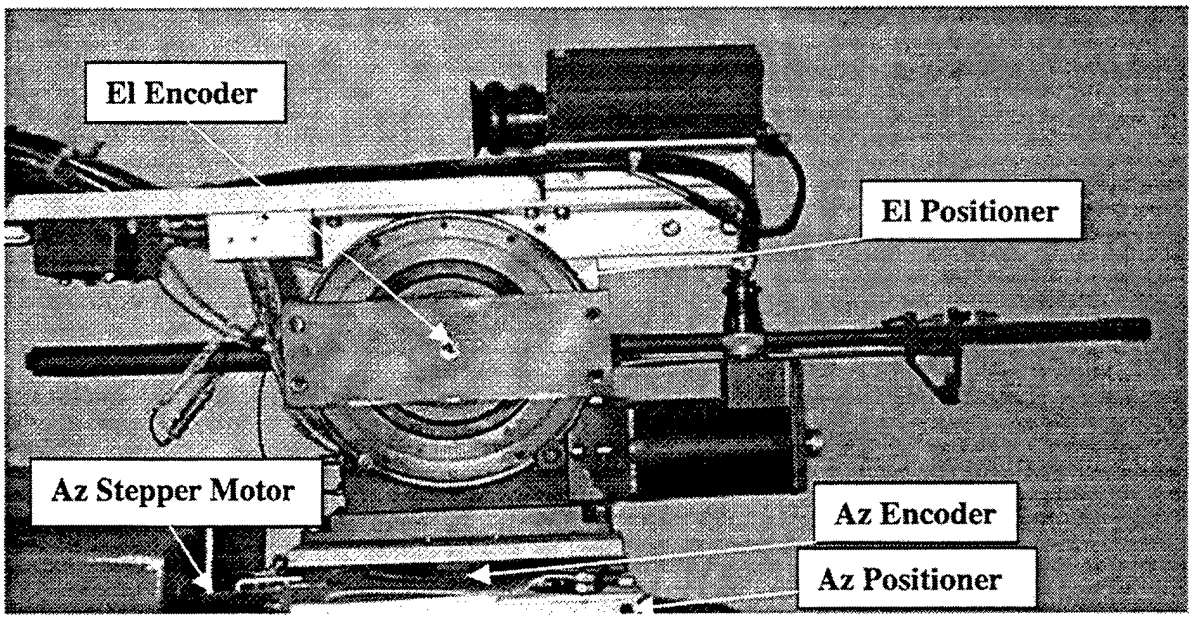

Figure 1. The IRT Applied to an M16A2 Rifle Firing From a Fast Attack Vehicle (Right Side View).

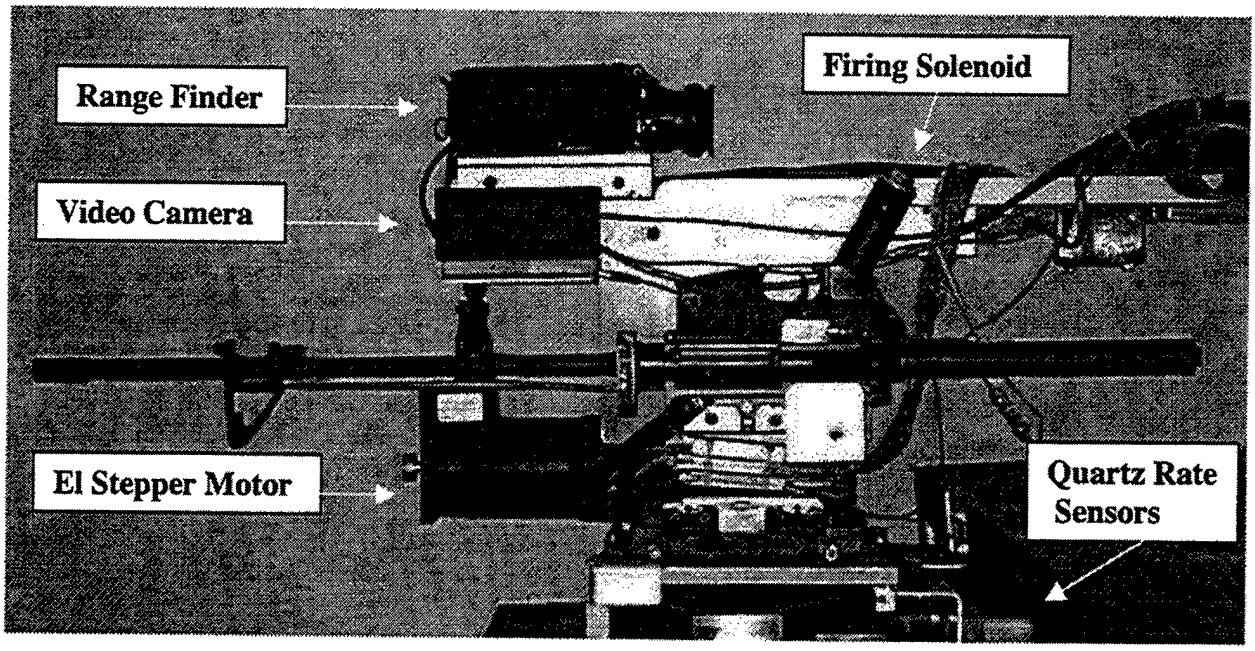

Figure 2. The IRT Applied to an M16A2 Rifle Firing From a Fast Attack Vehicle (Left Side View).

The 5.56-mm M16A2 rifle is mounted on two Aerotech (ART 312) rotary positioning stages. One stage is used for the elevation direction and the second stage is used for the azimuth direction. Both stages are driven by stepper motors that are powered by Dynacron (DM 8010) microstepping translators. Two Itek LSI Micro Series (μS/16/23K) shaft angle encoders measure the relative angular displacements between the weapon and the weapon platform in azimuth and elevation. The shaft angle encoders are mounted inside of the rotary positioning stages, and their output shafts are attached to the elevation and the azimuth axis of the weapon platform. Three Systron Donner quartz rate sensors (QRS-11-0010-101) mounted to the weapon platform measure the angular displacements of the weapon platform in elevation, azimuth, and roll. A simple wheel counter that was designed and built at ARL measures vehicle translation.

The initial range of the vehicle to the target is measured using a Helios range finder that is mounted just above the CCD camera block. The range finder is controlled remotely from inside the vehicle by switches on the remote control panel. Once the initial range to the target is put into the computer and the inertial reticle is accurately placed over the desired target, the target designator switch on the gunner's control panel is momentarily engaged. At this time, the initial

position of the target relative to the inertial reticle is determined in inertial coordinates. Theoretically, once the inertial reticle is positioned accurately over the target, it should stay there indefinitely. However, due to random walk in the quartz rate sensors, the inertial reticle will drift off from the target after several seconds. The inertial reticle can be easily repositioned over the target by slight movements of the joystick. If the drift becomes large enough to get outside of the range of the joystick correction, a switch on the gunner's control panel can be turned on, switching the joystick operation from the displacement mode to the velocity mode and giving it an endless correction range. A second switch on the gunner's control panel can also be engaged when the joystick is in velocity mode. This causes the stepper motors of the weapon positioner to move at a much faster rate, which facilitates getting the target in the field of view.

Control of firing the weapon is accomplished by means of an arm switch and a fire button that are mounted on the gunner's control panel. Once the arm switch is turned on and the gunner is satisfied with the position of the inertial reticle over the desired target, then the gunner depresses the fire button and holds it depressed to enable the electrical firing solenoid. Using the ballistic solution, the weapon automatically fires such that the projectile exit from the barrel occurs when the muzzle of the weapon is properly aligned on the target.

The integration of the quartz rate sensors signals, the reading of the wheel counter, the reading of the shaft angle encoders, the determination of the target position in inertial space, the control of the weapon positioner, the generation of the predictive fire control algorithm, and the firing of the weapon are accomplished by a small WinSystems 486 SLC computer and power supply. The generation of the electronic pointers is accomplished by a small 386 SX computer that is fed directly into the WinSystems 486 SLC computer.

The complete computer programs for both of the computers are presented in the Appendix. The gunner's control panel is shown in Figure 3. The video image as seen on the monitor in the gunner's control panel is shown in Figure 4. The range to the target is shown in the upper left corner. The diameter of the black target in the center of the video image is 20.3 cm.

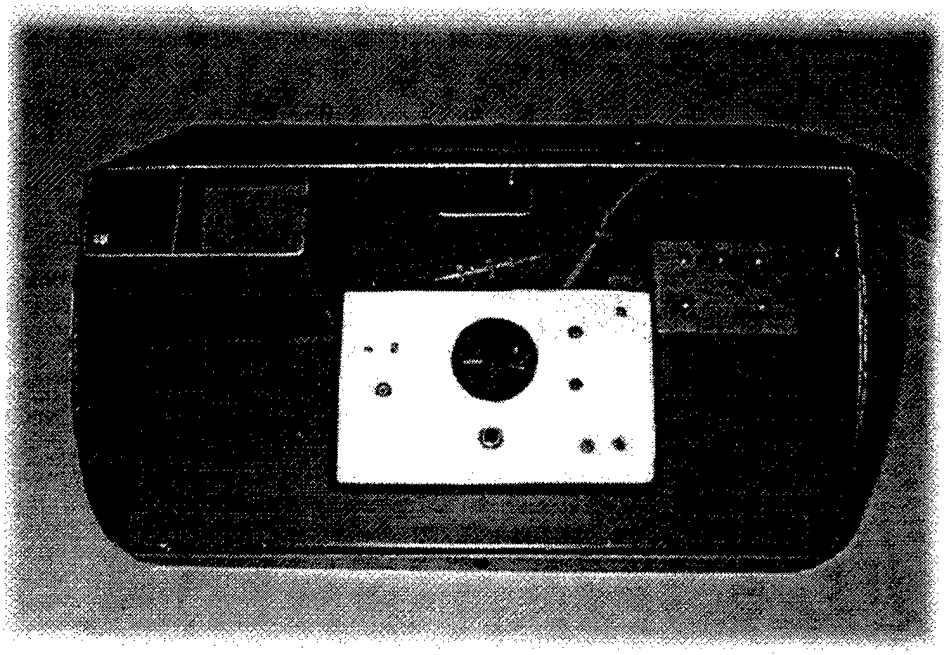

Figure 3. Remote Control Panel (Rests on Gunner's Lap).

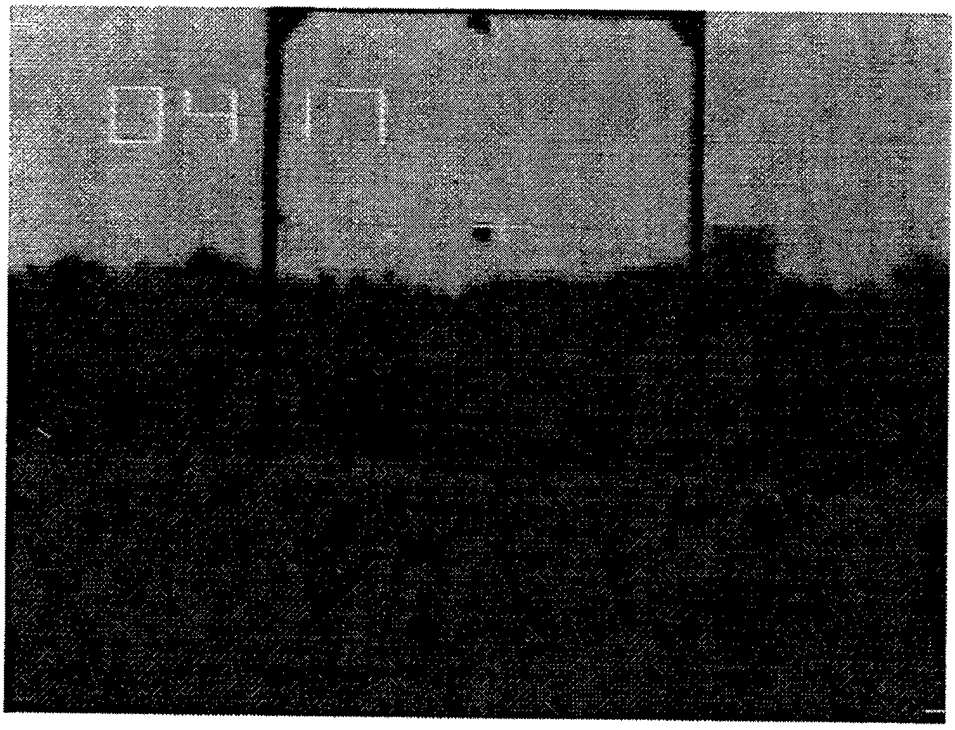

Figure 4. Video Image of 20.3-cm Target at 417 m (Monitor in Remote Control Panel).

4. Indoor Testing of the IRT Applied to a 5.56-mm M16A2 Rifle

Prior to any long-range testing of the IRT applied to a 5.56-mm M16A2 rifle firing from a fast attack vehicle, extensive short-range testing was done in the indoor range in Building 390 at ARL. Over 100 rounds were fired with the weapon positioner mounted to a rigid plate to determine if the weapon positioner, the video camera, the shaft angle encoders, and the quartz rate sensors could withstand the shock from firing. Accuracy measurements were also taken during the initial testing for each round fired. The accuracy acceptance specification for M855 ammunition, which was used in the initial testing, converts to an average standard deviation of .28 mil in both the elevation and the azimuth directions. The average standard deviations in the elevation and the azimuth directions for several 10-round groups fired with the weapon positioner mounted to a rigid plate were .26 mil and .24 mil, respectively. Since the average standard deviation for the experiments with the weapon positioner mounted to a rigid plate were essentially the same as the accuracy acceptance specification, it was felt that the IRT integrated with the 5.56-mm M16A2 rifle was achieving its maximum accuracy performance for these conditions and no further short-range indoor experiments were performed. There was also no noticeable damage to the weapon positioner, the video camera, the shaft angle encoders, or the quartz rate sensors after firing over 100 rounds.

5. Initial Long-Range Outdoor Testing of the IRT Applied to a 5.56-mm M16A2 Rifle Firing From a Fast Attack Vehicle

After the indoor testing was completed, initial long-range outdoor firing of the IRT applied to the 5.56-mm M16A2 IRT test bed was mounted on a fast attack vehicle and initial long-range outdoor firing experiments were done at the U.S. Army Aberdeen Test Center (ATC) H-Field test facility at the Edgewood Area of Aberdeen Proving Ground. Before any firings from a moving vehicle were done, firings from a stationary vehicle were made at a 400-m target. The average standard deviations in the elevation and the azimuth directions for several five-round groups of M855 ammunition fired semiautomatically by the gunner from inside the stationary

vehicle were .29 mil and .30 mil, respectively. Since the average standard deviations for the firings by the gunner from inside the stationary vehicle were essentially the same as the accuracy acceptance specification of .28 mil, it was felt that the IRT integrated with a 5.56-mm M16A2 rifle was achieving its maximum accuracy performance for these conditions and no further stationary long-range experiments were required, and firing-on-the-move experiments were initiated.

After completing the stationary vehicle firings at the 400-m target, the firings were repeated with the gunner firing from inside the vehicle while the vehicle was moving toward the target. Five-round groups were fired semiautomatically by the gunner at about 1-s intervals, while the vehicle was traveling at 16 kph down a gravel road toward a 400-m target. The average standard deviations in the elevation and the azimuth directions for several five-round groups of M855 ammunition fired semiautomatically by the gunner from inside the moving vehicle were .93 mil and .87 mil, respectively. Since the standard deviations were considerably higher than those obtained in the stationary vehicle firings, the firings were stopped to determine why the standard deviations were so much higher.

In reviewing the video tape of the inertial reticle taken during the firings from the moving vehicle, it was determined that a high-frequency oscillation of about 25 Hz was being transmitted from the frame of the fast attack vehicle into the weapon platform as the fast attack vehicle was traveling down the gravel road at 16 kph. The weapon controller and the IRT sensors easily handled the 25-Hz oscillations and held the reticle over the aim point, but the video image was moving so rapidly that multiple reticles appeared that made it very difficult to accurately hold the inertial reticle on the target.

The 25-Hz oscillation also caused a serious problem with the firing predictor, because the time interval from the firing pulse to the projectile exit from the gun barrel was 30 ms for the hammer-fired 5.56-mm M16A2 rifle. At 25 Hz, the firing predictor could not predict reliably out to 30 ms and there were many instances when the muzzle of the weapon was not pointing at the target when the projectile exited the gun barrel. To prevent the 25-Hz oscillation in the frame of the fast attack vehicle from being transmitted to the weapon platform, an inexpensive isolation

mount was designed and built and placed between the frame of the fast attack vehicle and the weapon platform. By using two roller bearings, two swivel bearings, several light springs, three oil-filled dashpots, and extra weights, the frequency of the weapon platform was reduced to about 3 Hz in the elevation, the azimuth, and the roll directions when the fast attack vehicle was traveling at 16 kph down the gravel road. At 3 Hz, there were no multiple inertial reticles on the video image and the inertial reticle could easily be held on target. The firing predictor was also easily able to predict reliably out to 30 ms. The isolation mount can be seen in Figure 5 between the frame of the fast attack vehicle and the weapon platform.

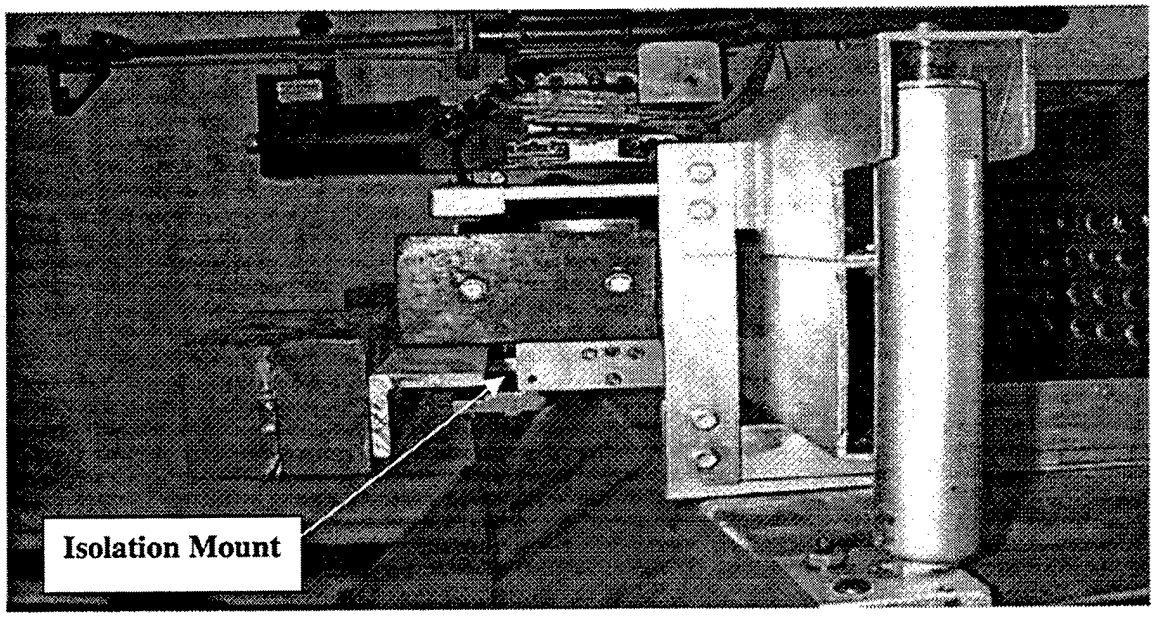

Figure 5. The Isolation Mount Between the Frame of the Fast Attack Vehicle and the Weapon Platform.

6. Final Long-Range Outdoor Testing of the IRT Applied to a 5.56-mm M16A2 Rifle Firing From a Fast Attack Vehicle

After the initial long-range outdoor testing was completed and the isolation mount was installed and tested extensively in nonfiring runs at 16 kph down a gravel road, final long-range outdoor testing of the IRT applied to an M16A2 rifle firing from a fast attack vehicle was done at

the H-Field test facility at the Edgewood Area of Aberdeen Proving Ground. The firing experiments from a moving vehicle done in the initial long-range outdoor testing of the IRT were repeated on the same firing range. Ten rounds were fired semiautomatically by the gunner from inside the vehicle at about 1-s intervals while the vehicle was moving at 16 kph down the same gravel road toward the 400-m target. The average standard deviations in the elevation and the azimuth directions for several 10-round groups of M855 ammunition fired semiautomatically by the gunner from inside the moving vehicle were .47 mil and .43 mil, respectively. The extreme spread for the firings was 43 cm.

In reviewing the videotape of the inertial reticle taken during the firing experiments from the moving vehicle, it was determined that the weapon platform was oscillating at a frequency of about 3 Hz. A check of the firing time data also taken during the firing experiments showed that there were no instances when the muzzle of the weapon was not pointing at the target when the projectile exited the gun barrel. Since the average standard deviations for the firings by the gunner from inside the moving vehicle were only slightly higher than those fired from the stationary vehicle, it was felt that the IRT applied to an M16A2 rifle firing from a fast attack vehicle was achieving its optimum performance in accuracy for this scenario. Once the firings from the moving vehicle while driving toward the 400-m target were completed, the target was placed 400-m off to the side of the vehicle and the firing experiments were repeated with the gunner firing from inside the moving vehicle and the weapon pointing over the right and left sides of the vehicle while it traversed parallel to the target along a gravel road at 16 kph. The IRT held the inertial reticle on the target and put in the correct amount of lag angle so that the projectiles hit on target. The average standard deviations for the elevation and the azimuth directions for several 10-round groups of M855 ammunition fired semiautomatically by the gunner from inside the moving vehicle were .48 mil and .50 mil, respectively. The extreme spread for the firings was 48 cm. The average standard deviations were essentially the same as those for the previous firings for the vehicle traveling straight toward the target.

To check out the capability of the IRT to fire at a moving target, if data on the movement of the target were available to the IRT computer, the first firing experiments of firing at a 400-m

target while the vehicle was traveling along a gravel road at 16 kph toward the target were repeated. However, in the new experiments, the target was moving left to right at 16 kph. Ten-round groups were fired semiautomatically by the gunner from inside the vehicle at about 1-s intervals while the vehicle was moving down the gravel road at 16 kph toward the 400-m target. The IRT held the inertial reticle on the target and put in the correct amount of lead angle so that the projectiles hit on target. The average standard deviations in the elevation and the azimuth directions for several 10-round groups of M855 ammunition fired semiautomatically by the gunner from inside the moving vehicle were .40 mil and .45 mil, respectively. The extreme spread for the firings was 46 cm. These average standard deviations were essentially the same as those for the previous firings for the vehicle traveling straight toward the target.

7. Conclusions

(1) The IRT applied to an M16A2 rifle firing from over the front of a fast attack vehicle improved the accuracy to such an extent that a test engineer was able to keep 10-round groups of M855 ammunition to within a 43-cm circle, which was centered on the target, while firing at the rate of about 60 rd/min from inside the vehicle while it was moving at 16 kph toward a 400-m target.

(2) The IRT applied to an M16A2 rifle firing from over the side of a fast attack vehicle improved the accuracy to such an extent that a test engineer was able to keep 10-round groups of M855 ammunition to within a 48-cm circle, which was centered on the target, while firing at the rate of about 60 rd/min from inside the vehicle while it was moving at 16 kph parallel to a 400-m target.

(3) The IRT applied to an M16A2 rifle firing from over the front of a fast attack vehicle improved the accuracy to such an extent that a test engineer was able to keep 10-round groups of M855 ammunition to within a 46-cm circle, which was centered on the target, while firing at the rate of about 60 rd/min from inside the vehicle while it was moving at 16 kph toward a 400-m target that was moving left to right at 16 kph.

INTENTIONALLY LEFT BLANK.

Appendix:

Computer Programs*

* Program Charlie4 is the computer program for the 386 SX computer. Program Z is the computer program for the WinSystem 486 SLC computer.

INTENTIONALLY LEFT BLANK.

```
Program Charlie4; {May 3rd, 1994, 09:54 A.M., November 3, 1994}
  uses                    {Video Charlie}
    graph,   crt;
  type a10 = array[0..9] of word;

    var       A           :a10;
          Gd, Gm           :integer;
          p0, p1           :boolean;
          ccc,ddd,i,j,k,l  :byte;
          vvv,www          :word;
    x11,x12,x21,x22,y11,y12,y21,y22:word;
    r0,r1,q0,a11,b11:word;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
Procedure Initialize;
begin
  asm
    mov dx,74bh
    mov al, 09bh  {Ports A, B and C all input}
    out dx,al
  end;
;
  x11:=0; x12:=0; y11:=0; y12:=0; x21:=0; x22:=0; y21:=0; y22:=0;
  a11:=320; b11:=100; a[0]:=0; a[1]:=0; a[2]:=0; a[3]:=0;
end;


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
Procedure Plot_It (x,y:byte; var v:byte);
    begin

        SetColor(Black);

        r0:=x;r1:=x+7;
        q0:=y;
        line(r0,q0,r1,q0);

        r0:=x;r1:=x+7;
        q0:=y+1;
        line(r0,q0,r1,q0);

         r0:=x;r1:=x+7;
        q0:=y+2;
        line(r0,q0,r1,q0);
```

```
        r0:=x;r1:=x+7;
        q0:=y+3;
        line(r0,q0,r1,q0);

        r0:=x;r1:=x+7;
        q0:=y+4;
        line(r0,q0,r1,q0);

        r0:=x;r1:=x+7;
        q0:=y+5;
        line(r0,q0,r1,q0);

        r0:=x;r1:=x+7;
        q0:=y+6;
        line(r0,q0,r1,q0);

        setcolor(white);
      if v < 5 then
       begin {0-4}
        case v of
          0:outtextxy(x,y,'0');
          1:outtextxy(x,y,'1');
          2:outtextxy(x,y,'2');
          3:outtextxy(x,y,'3');
          4:outtextxy(x,y,'4');
        end
       end
      else
       begin {5-9}
        case v of
          5:outtextxy(x,y,'5');
          6:outtextxy(x,y,'6');
          7:outtextxy(x,y,'7');
          8:outtextxy(x,y,'8');
          9:outtextxy(x,y,'9');
        end
       end
    end;


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
Procedure GetData(var ddd,ccc:byte; var vvv,www:word; var p0,p1:boolean);
begin
 {sxxx xxxx xccc cddd  dddd dddd}
```

```
asm  {$G+}
     mov dx,749h
@0:              in al,dx      ;  mov cl,al   {xccc cddd > cl}
     dec dx    ;  in al,dx      ;  mov ch,al   {dddd dddd > ch}
     inc dx    ;  in al,dx                     {xccc dddd > al}
     and al,cl  ;  and al,080h   ; jz @0

     les bx,DDD
     mov al,cl                              {xccc cddd > al}
     ror al,3                               {dddx cccc > al}
     and al,0fh                             {0000 cccc > al}
     mov es:[bx],al                         {0000 cccc > DDD}
;
     les bx,VVV  ;  mov es:[bx],ch                {dddd dddd > VVV}
     mov al,cl   ;  and al,07h    ;               {0000 0ddd > al}
     mov es:[bx+1],al                           {0000 0ddd > vvv+1}
;
     les bx,CCC     ; {Bit 7 of port 74ah is connected to}
     mov dx,074ah   ; {the external video sync.}
     in al,dx       ; {No other bits are used.}
     mov es:[bx],al
end;

  A[DDD] := VVV;

begin

p1 := p0;
if ((CCC and 128) > 0) then  p0 := true else p0 := false;

if ( (p0 = true) and (p1 = false)  ) then
 begin

   begin
      vvv := A[4];
      i := vvv div 1000; vvv := vvv - i*1000; plot_it(10,10,i);
      j := vvv div  100; vvv := vvv - j* 100; plot_it(20,10,j);
      k := vvv div   10; vvv := vvv - k*  10; plot_it(30,10,k);
      l := vvv;                               plot_it(40,10,l);
   end;


   begin
      SetColor(Black);
      line(x11,y11,x12,y12);
```

```
    line(x21,y21,x22,y22);
    line(x21-2,y21,x22-2,y22);
    line(x21+2,y21,x22+2,y22);

    VVV := A[2];
    if vvv > 638 then vvv := 638;
    x11 := vvv - 20; if x11<  1 then x11:=  1;
    x12 := vvv + 20; if x12> 638 then x12:= 638;
    x21 := vvv; x22 := vvv;

    WWW := A[3];
    if www > 198 then www := 198;
    y21 := www -  8; if y21<  1 then y21:=  1;
    y22 := www +  8; if y22> 198 then y22:= 198;
    y11 := www; y12 := www;

    SetColor(White);
    line(x11,y11,x12,y12);
    line(x21,y21,x22,y22);
    line(x21-2,y21,x22-2,y22);
    line(x21+2,y21,x22+2,y22)
end;

begin
    SetColor(Black);
    line(a11-20,b11-8,a11+20,b11-8);
     line(a11-20,b11+8,a11+8,b11+8);

    VVV := A[0];
    if vvv > 630 then vvv := 630;
    if vvv <  8 then vvv :=  8;
    a11 := vvv;

    WWW := A[1];
    if www > 190 then www := 190;
    if www <  8 then www :=  8;
    b11 := www;
    {if(ddd and 2 = 0}

    SetColor(White);
    line(a11-20,b11-8,a11+20,b11-8);
     line(a11-20,b11+8,a11+8,b11+8);

end;
```

```
 end;  {if ( (p0 = true) and (p1 = false)  )}
 end;
end;  {procedure}


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
  begin  {Main}

    Gd:=detect;
    Initgraph (Gd, Gm, 'c:\tp\bgi');
    if graphresult <> grOk then
      begin
        writeln('Cannot file graphics files.  Press any key to continue.');
        readln; halt (1);
      end;


(*------ initial values ------*)

Initialize;     p0:= true;

while keypressed = false do  GetData(ddd,ccc,vvv,www,p0,p1);
     closegraph;
 end.
```

```
Program Z;    {September 14, 1992, June 21, 1993, March 24, 1994}
{N+}          {March 31, 1994 - Predict 40 ms, cycle 2 ms}
              {April 25, 1994 - used with video reticle genrator}
              {February 7, 1995, February 21, 1995}
              {April 30, 1996, May 8, 1996}
              {May 14, 1996 used with GPS}
  {August 9, 1996 - new processor board (33MhZ), new dt,  new time: 6.0ms}
  {August 9, 1996 - new delays for the Laser Range Finder}
  {October  1996 - Rewritten and updated.  Includes wheel geometry}
  {January  1997 - New predictor - outputted to DAC ports/with firing pulse}
  {November 4,1997 - Camera stabilization}
  {November 20,1997- New camera stabilization}
   {BFCs denotes Body Fixed Coordinates, ICs denotes Inertial Coordinates}

Uses  Graph,Crt;          {Port assignments: 050h to 057h}

Const B_V=20; B_H=30;   B_V2=B_V div 2; B_H2=B_H div 2;
Const bit0=0;bit1=1;bit2=4;bit3=8;bit4=16;bit5=32;bit6=64;bit7=128;
Const  EarthRadius = 6369537.34;   saeo_c = 1750; saep_c= -440;
     WheelToSight_X = 1.2; WheelToSight_Y = 1.0; WheelToSight_Z = 2.0;{BFC}
     ConversionToRadians = 2*pi/65536; NoOfConversionsPerSecond = 3500;
     Coeff1=1.05*(10/9)*(pi/2)/131071; {100 degrees per second > 2**17-1}
     ANG=0.04997558594;
Type
     seal  = double;
     r33   = array[1..3,1..3] of seal; i88 = array[0..7,0..7] of byte;
     r8    = array[0..7] of seal;
     i8    = array[0..7] of byte;
     r3    = array[1..3] of seal;
     ar128b = array[1..150] of byte;
     i3    = array[1..3]   of integer;
     a10   = array[0..10] of seal;  var r,v:a10;
Var
    ww,swz,tee,PortC0,PortC1,flop      : byte;
    s                          : ar128b; {Supports the GPS}
  c_azi,c_ele,      CosO,SinO,CosP,SInP : Seal;
  xc,yc, gd, gm, i,m ,e              : integer;
Fir,FirP,Error, GPS, WC, Counter       : boolean;
abc:boolean;
                          nnn   : i8;
tau,  we0,we1,we2,we3,we4,we5,we6      :seal;
  Range, Azi0,Azi1,Ele0,Ele1,Temp,Slew : seal;
  B1,B2, dt, sum1,  sum2, wx, wy,  wz  : seal; {Body fixed angular rates}
  C_Joy_O, C_Joy_P, P1, P2,temp2,  SE  : seal;
  Port7Bit6, Port7In, BI0,BI1,Bib, sam : byte;
```

```
  channel,   n3,  n2,  n1,  n0 : byte; {Raw sensor inputs}
 port6, Port9_In, n7,  n6,  n5,  n4 : byte; {Raw sensor inputs}
 PortB_In,P7InP, j, k, cnt, P7In, PCIn : byte; {couters and flag}
 Looper,          Delay,  wrd    : Word;
                        nn : i88;   {Raw sensor inputs}
         we7,   LLL,  Count   : word;
 wappp,  wapp,wap                : seal;
 weppp,  wepp,wep                : seal;
 ii_c_azi, con1,con2,alpha        : seal;
 Dist,  Temp_word,app,bpp : word;
    SaeO, SaeP,SaeO_, SaeP_       : seal;
 IntO,IntP, JoyO, JoyP, Speed,err,zse  : seal;
vxx, vyy, vzz,dxx,dyy,dzz,dx,dy,dz    : seal;
                    a      : r33;  {Transformation matrix}
i_c_ele,i_c_azi, xi,  et,  ze,  ch   : seal; {Quaterian vriables}
c_a_drift,c_e_drift, xidp,etdp,zedp,chdp: seal;  {Predicted derivatives}
 D_Az0,D_El0,  xid0, etd0, zed0, chd0  : seal;  {Previous derivatives}
 rr, ss,  tt,  xx, yy, zz,      dt2  : seal;  {Normalized time step}
 rrr,sss,ttt,xxx,yyy,zzz,D_Az1,D_El1    : seal;  {Body fixed angular rates}
 xa,ya,za,xb,yb,zb                : seal;  {Body fixed linear accelerations}
 z0,z1,z2,z3,z4,d0,d1,d2,d3,d4,fff     :boolean;

 c_ele_f,c_azi_f, azi_lim, ele_lim     : seal;
 TimeOfFlight,xw, yw, zw, t,azza,azze  : seal;  {Defines point 1, previous}
 DriftX, DriftY, DriftZ, sum         : seal;
 JoyP_,JoyO_,o,oo,ooo,p,pp,ppp,ao,ap   : seal;
 YO9,YP9,YO,YP,SO,SP,john,jane        : seal;
 aa,bb,cc,dd,ee,aabb,ccdd, col       : r3;
 the,phi,alt, zzzz                 : real;
 indx                        : i3;



Procedure PutP3( var A:Byte); begin asm  mov dx,233h; les bx, A; mov al,es:[bx];  out dx,al
end end;
Procedure PutP6( var A:Byte); begin asm  mov dx,236h; les bx, A; mov al,es:[bx];  out dx,al
end end;
Procedure D; var  i,j: byte;
        begin for i := 0 to 255 do for j:= 0 to 255 do begin  end end;
Procedure CX100_Mode; var P3:byte;begin P3:=$47; PutP3(P3); D end;
Procedure Dis_Ovl; var P6:byte;   begin P6:=$27; PutP6(P6); D end;
Procedure Ovr_Ena; var P6:byte;   begin P6:=$29; PutP6(P6); D end;
Procedure Dis_Int; var P6:byte;   begin P6:=$10; PutP6(P6); D end;
Procedure High_Res;var P6:byte;   begin P6:=$0a; PutP6(P6); D end;
```

```
Procedure Set_Pix(var A,B:word);
begin asm {$G+}
 les bx,B
 mov cx,es:[bx] {cx: 0 0 0 0  0 0 0 b8  b7 b6 b5 b4  b3 b2 b1 b0}
 les bx,A
 mov dx,es:[bx] {dx: 0 0 0 0  0 0 0 a8  a7 a6 a5 a4  a3 a2 a1 a0}
 rol dx,1      {dx: 0 0 0 0  0 0 a8 a7  a6 a5 a4 a3  a2 a1 a0 0}
 xor al,al
 mov ah,dl     {ax:a6 a5 a4 a3  a2 a1 a0 0  0 0 0 0  0 0 0 0}
 add cx,ax     {cx:a6 a5 a4 a3  a2 a1 a0 b8  b7 b6 b5 b4  b3 b2 b1 b0}
 mov al,dh     {al:                0 0 0 0  0 0 a8 a7}
 and al,03h    {al:                0 0 0 0  0 0 a8 a7}

 mov dx,236h   {Output the Page_Select bits to Port 6}
 out dx,al     {Output to P0, program the two "page" bits, P0.1, P0.0}
{D}                          mov al,1;   @0: dec al; jnz @0
 mov ax,0d000h {ax: 1 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0}
 mov es,ax     {es: 0 0 0 0  1 0 1 0  0 0 0 0  0 0 0 0}
mov al,1bh; out dx,al {Ram Enable}
{D}                          mov al,1;   @1: dec al; jnz @1
mov al, 29h; out dx,al {OverLay_Enable}
{D}                          mov al,1;   @2: dec al; jnz @2
 mov bx,cx     {bx:a6 a5 a4 a3  a2 a1 a0 b8  b7 b6 b5 b4  b3 b2 b1 b0}
 mov al, 00h     {ax: * * * * * * * *}
 mov es:[bx],al {Move pixel value to video memory}
{D}                          mov al,1;   @3: dec al; jnz @3
 mov al,es:[bx]  {write to video ram}
 mov al,1ah; out dx,al  {Ram Disable}
end end;

Procedure ReSet_Pix(var A,B:word);
begin asm {$G+}
 les bx,B; mov cx,es:[bx]; les bx,A; mov dx,es:[bx]; rol dx,1; xor al,al
 mov ah,dl; add cx,ax; mov al,dh; and al,03h;
 mov dx,236h; out dx,al; mov al,1;   @0: dec al; jnz @0
 mov ax,0d000h; mov es,ax; mov al,1bh; out dx,al; mov al,1;  @1: dec al; jnz @1
 mov al, 29h; out dx,al; mov al,1;   @2: dec al; jnz @2
 mov bx,cx; mov al, 0fh  {0,1 or 15}
 mov es:[bx],al; mov al,1;   @3: dec al; jnz @3
 mov al,es:[bx];  mov al,1ah; out dx,al
end end;
```

```
Procedure LineV( x0, y0,n:word; color:boolean);
var i :word;
begin
   for i := 0 to n do
      begin
         if color then Set_Pix(x0,y0) else ReSet_Pix(x0,y0);
         y0:=y0+1;
      end;
  end;

Procedure LineH( x0, y0,n:word; color:boolean);
var i :word;
begin
   for i := 0 to n do
      begin
         if color then Set_Pix(x0,y0) else ReSet_Pix(x0,y0);
         x0:=x0+1;
      end;
  end;

Procedure G_Init;
var i:word;
begin
CX100_Mode; {Places boared in the CX100 mode}
Dis_Ovl; {Display_Overlay}
Ovr_Ena; {Overlay_Enable}
Dis_Int; {Disable Interrupts}
High_Res; {Enables High Resolution}
for i:= 0 to 511 do  LineH(i,0,511,false);
end;

Procedure G_Set( x0,y0:word;c:boolean);
begin
LineV(x0,   y0,     B_V,c);
LineH(x0,   y0,     B_H,c);
LineH(x0,   y0+B_V,  B_H,c);
LineV(x0+B_H,y0,     B_V,c);
end;


 PROCEDURE LuDcmp( n :integer; VAR   a: r33;  VAR indx :i3; VAR  d :seal);
 CONST tiny=1.0e-20;

 VAR k,j,imax,i:integer;
 sum,dum,big:seal;
```

```
      vv:r3;
 BEGIN
 ;
 d:=1.0 ;
 FOR i:=1 TO n DO BEGIN
  big:=0.0;
   FOR j:= 1 TO n DO  IF (abs(a[i,j]) > big) THEN  big:=abs(a[i,j]);
   IF (big=0.0) THEN BEGIN     END; {if}
   vv[i]:=1.0/big;
 END; {for i}
;
 FOR j:= 1 TO n DO BEGIN
  IF (j>1) THEN BEGIN
   FOR i:=1 TO j-1 DO BEGIN
    sum:=a[i,j];
    IF (i>1) THEN BEGIN
      FOR k:= 1 TO i-1 DO sum :=sum-a[i,k]*a[k,j];  a[i,j] := sum;
    END {if i}
   END {for i}
  END; {if j}
;

 big:=0.0;
 FOR i:=j TO n DO BEGIN
    sum:= a[i,j];
    IF (j>1) THEN BEGIN
      FOR k:=1 TO j-1 DO sum:=sum-a[i,k]*a[k,j]; a[i,j]:= sum;
    END; {if j}
  dum:= vv[i]*abs(sum);
  IF (dum>big) THEN BEGIN  big:=dum; imax :=i END {if dum}
 END; {for i}
;
 IF (j<> imax) THEN
    BEGIN
      FOR k:= 1 TO n DO BEGIN
         dum:=a[imax,k]; a[imax,k]:=a[j,k]; a[j,k]:=dum;
      END; {for k}
     d:=-d; vv[imax]:=vv[j];
     END;  {if j}
;
  indx[j]:= imax;
  IF(j<>n) THEN
    BEGIN
      IF(a[j,j]=0.0) THEN a[j,j]:=tiny;  dum :=1.0/a[j,j];
      FOR i := j+1 TO n DO a[i,j]:=a[i,j]*dum;
```

```
    END {if j}
   END; {for j}
;
  IF(a[n,n] =0.0) THEN a[n,n] := tiny;
 END; {proc}


Procedure LuBkSb(n :integer; VAR indx :i3; VAR b :r3; VAR a :r33);
VAR j,ip,ii,i:integer;
  sum:seal;
BEGIN
  ii:=0;
  FOR i:=1 TO n DO BEGIN
    ip:=indx[i]; sum:=b[ip]; b[ip]:=b[i];
    IF (ii <>  0) THEN
       BEGIN
          FOR  j:= ii TO i-1 DO sum:=sum-a[i,j]*b[j];
       END {if ii}
    ELSE    IF (sum <> 0.0) THEN        II:=  i;
    b[i]:=sum;
  END; {for i}
  FOR i:= n DOWNTO 1 DO BEGIN
    sum := b[i];
    IF (i<n) THEN FOR j:=i+1 TO n DO sum := sum -a[i,j]*b[j];
    b[i]:= sum/a[i,i];
  END {for i}
END; {procedure}


  Procedure Mat_Inv(var a,y:r33); {Generates the inverse matrix of A in Y}
  var i,j,n:integer;   d:seal;    {The matrix A is destroyed}
  begin
  n:=3;
  LuDcmp(n,a,indx,d);
  for j := 1 to n do
   begin
    for i := 1 to n do  col[i]:=0 ;
    col[j]:=1.0;
    LuBkSb(n,indx,col,a);
    for i:= 1 to n do  y[i,j]:=col[i];
   end;
  end;
```

```
Procedure MATMAT ( var c:r33;  a, b:r33);
var i ,j,k:word; sum:seal;
begin
 for i :=1  to 3 do   for k := 1 to 3 do
 begin sum:=0;for j:=1 to 3 do  sum:=sum+a[i,j]*b[j,k]; c[i,k] := sum end;
end;


Procedure MATMUL(var a,b:r3;var c:r33);
var i,j:word; sum:seal;
begin for i:= 1 to 3 do begin sum := 0;
  for j := 1 to 3 do sum := sum + b[j]*c[i,j];
  a[i] := sum end end;


 Procedure MATMULInv(var a,b:r3;var c:r33);
 var i,j:word; sum:seal;
 begin for i:= 1 to 3 do begin sum := 0;
  for j := 1 to 3 do sum := sum + b[j]*c[j,i];
  a[i] := sum end end;


Procedure VDIF(var a,b,c:r3);
var i:word;begin for i:=1 to 3 do a[i]:=b[i]-c[i] end;


Procedure CMULT( var a,b,c:r3);
begin a[1]:=b[2]*c[3]-b[3]*c[2]; a[2]:=b[3]*c[1]-b[1]*c[3];
    a[3]:=b[1]*c[2]-b[2]*c[1]  end;


 Procedure Norm(var a:r3);
 var temp:seal; i :word;
  begin temp := 0;
  for i := 1 to 3 do temp := temp + sqr(a[i]); temp := sqrt(temp);
  for i := 1 to 3 do a[i]:=a[i]/temp end;


 Procedure DotProd(var r:seal;a,b:r3);
 begin r := a[1]*b[1]+a[2]*b[2]+a[3]*b[3]; end;
```

```
Function ATan (y,x:real):real;
var  u:real;
begin
 if ( (x=0) and (y=0) ) then atan:=0.0 else
 begin
  if (abs(x) < abs(y))  then
  begin {abs(x) <  abs(y)}
   u:=arctan(abs(x/y));
   if x<0 then
    begin {x<0} if y>0 then atan:=pi/2+u else atan:=-pi/2-u end else
    begin {x>0} if y>0 then atan:=pi/2-u else atan:=-pi/2+u end
  end else
  begin {abs(x) >= abs(y)}
   u:= arctan(abs(y/x));
   if x<0 then
    begin {x<0} if y>0 then atan:=pi  -u else atan:= -pi +u end else
    begin {x>0} if y>0 then atan :=    u else atan:=     -u end
  end
 end
end;


Procedure Mat(var xi,et,ze,ch:seal;  var a:r33);
var  ze2, et2, xi2, ch2, ze_et, ze_xi, ze_ch, xi_et, et_ch, xi_ch:seal;
begin  {calculates elements of the transformation matrix}
 ze2 := ze*ze; xi2 := xi*xi; et2 := et*et; ch2 := ch*ch;  et_ch:=et*ch;
 ze_et:=ze*et; ze_xi:=ze*xi; ze_ch:=ze*ch; xi_et:=et*xi;  xi_ch:=xi*ch;

 a[1,1]:=xi2-et2-ze2+ch2;a[1,2]:= 2*(xi_et+ze_ch);a[1,3]:= 2*(ze_xi-et_ch);
 a[2,1]:=2*(xi_et-ze_ch);a[2,2]:=-xi2+et2-ze2+ch2;a[2,3]:= 2*(ze_et+xi_ch);
 a[3,1]:=2*(ze_xi+et_ch);a[3,2]:= 2*(ze_et-xi_ch);a[3,3]:=-xi2-et2+ze2+ch2
end; {procedure}


Procedure Mat_Der(var xi, et, ze, ch, w1, w2, w3, xi_, et_, ze_, ch_:seal);
begin xi_:=( ch*w1 - ze*w2 + et*w3)/2; et_:=( ze*w1 + ch*w2 - xi*w3)/2;
    ze_:=(-et*w1 + xi*w2 + ch*w3)/2; ch_:=(-xi*w1 - et*w2 - ze*w3)/2 end;


Procedure ICsToBFCs(var x, y, z, x_, y_, z_ :seal);
begin  x:=x_*a[1,1]+y_*a[1,2]+z_*a[1,3];
     y:=x_*a[2,1]+y_*a[2,2]+z_*a[2,3];
     z:=x_*a[3,1]+y_*a[3,2]+z_*a[3,3];  end;
```

```
Procedure BFCsToICs (var x_,y_,z_:seal; x,y,z:seal);
begin  x_:=x * a[1,1] + y * a[2,1] + z * a[3,1];
       y_:=x * a[1,2] + y * a[2,2] + z * a[3,2];
       z_:=x * a[1,3] + y * a[2,3] + z * a[3,3] end;


Procedure Initialize_Q (var xi, et, ze, ch :seal);
begin xi:=0; et:=0; ze:=0; ch:=1; end;


Procedure Integrate(var a,b,c,d, u0,u1, v0,v1, w0,w1, x0,x1:seal);
begin a:=a+(u0+u1)*dt2;b:=b+(v0+v1)*dt2;c:=c+(w0+w1)*dt2;d:=d+(x0+x1)*dt2
end;


Procedure Up_Date(var a,b,c,d,e,f,g,h:seal);begin a:=b;c:=d;e:=f;g:=h end;


Procedure Normalize(var a,b,c,d:seal);
var sum:seal;   begin   {Normalize the Quaterion coefficients}
            sum:= sqrt( sqr(a) + sqr(b) + sqr(c) + sqr(d) );
            a:=a/sum; b:=b/sum; c:=c/sum; d:=d/sum end;


Procedure Step(var channel:byte; var value:word; var sign:boolean);
begin
  if channel = 0 then value := value + 32768;
  if sign = true then value := value + 16384;
  asm   {$G+}     {sccc cddd dddd dddd}
   mov dx,0300h;
   les bx,value;    mov ax,es:[bx];    {ah:dddd dddd, al:dddd dddd}
   out dx,al;       {dddd dddd}          {Bits 0 - 7 > 300h}
   mov dx,0302h;              {302h > dx}
   mov al,ah
   out dx,al
  end;
end;


Procedure Stepper_Driver( Channel:byte; var W:seal; Y0:seal; var S0:seal);
const alpha = 2000; beta = 100;
var sign:boolean;   Wp,a,b:seal;  wrd:word;
begin
```

```
  WP := W;
  W  :={ W +} alpha*y0*dt + beta*S0;
  if W > 0.33 then W := 0.33; if W < -0.33 then W := -0.33;
  if ( (P7In and 1) = 0) then W := 0;

  { Bound W. }
  if W <= Wp then
  begin if W < (Wp-0.005) then W := Wp-0.005 end
  else
  begin if W > (Wp+0.005) then W := Wp+0.005 end;

  a := W;
  if a < 0
  then begin sign := false; a := -a  end
  else begin sign := true;          end;


  if a > 0.00006 then  b := 1/a else b := 17000;
  wrd := trunc(b);
  if  wrd > 16383 then wrd := 16383;

  Step (channel, Wrd,sign);
end;



 Procedure GetPort7( var A:Byte);
 begin asm {$G+} mov dx,0331h; in al,dx; les bx, A; mov es:[bx],al end end;



(*
  Procedure GetPortC( var A:Byte);
  begin asm {$G+} mov dx,019Ch; in al,dx; les bx, A; mov es:[bx],al end end;
*)
  Procedure GetPortB( var A:Byte);
  begin asm {$G+} mov dx,0331h; in al,dx; les bx, A; mov es:[bx],al end end;



 Procedure SetPort6_1( var A:Byte);
 begin asm  {$G+} mov dx,0301h; les bx, A; mov al,es:[bx];  { (A) > al}
            or al,bit1;   mov es:[bx],al;  out dx,al end end;
```

```
Procedure ResetPort6_1( var A:Byte);
begin asm {$G+} mov dx,0301h; les bx, A; mov al,es:[bx];
     and al,255-bit1; mov es:[bx],al; out dx,al end  end;


Procedure SetPort6_7( var A:Byte);
begin asm {$G+} mov dx,0335h; les bx,A; mov al,es:[bx];
     or al,bit7;  mov es:[bx],al; out dx, al end end;


Procedure ResetPort6_7( var A:Byte);
begin asm {$G+} mov dx,0335h; les bx,A; mov al,es:[bx];
     and al,255-bit7; mov es:[bx],al; out dx,al end end;


Procedure SetPort6_0(var A:Byte);
begin asm {$G+} mov dx,0301h; les bx,A; mov al,es:[bx]; { (A) > al}
  or al,bit0; mov es:[bx],al; out dx,al end end;


Procedure ResetPort6_0(var A:Byte);
begin asm {$G+} mov dx,0301h; les bx, A; mov al,es:[bx];
  and al,255 - bit0; mov es:[bx],al; out dx,al end end;


Procedure Convert(var s:real; var nn:i8);
begin
{real: s:1   f:39    e:8. v := (-1)**s*2**(e-129)*(1.f).  if e=0 then v:=0}
{     b47  b46-b8  b7-b0  }
   asm          {MSByte  ah, al, ch, cl  LSByte}
     {$G+}
   les bx,nn
   mov al,es:[bx+4]    { al: s.in m.2 m.1 m.0  x   x  b25  b24}
   and al,3            { al:  0  0  0  0  0  0  b25  b24}

   mov dl,es:[bx+5]    { dh: s.in m.2 m.1 m.0  x  x  b27  b26}
   and dl,3            { dh:  0  0  0  0  0  0 b27  b26}
   rol dl,2
   or al,dl           { al:  0  0  0  0 b27 b26 b25 b24}

   mov dl,es:[bx+6]    { dl: s.in m.2 m.1 m.0  x  x  b29  b28}
   and dl,3
```

```
   rol dl,4
   or al,dl

   mov dl,es:[bx+7]   { dl: s.in m.2 m.1 m.0  x  x  b31  b30}
   and dl,3
   rol dl,6
   or al,dl          {al:  b31 b30 b29 b28 b27 b26 b25 b24}

   mov cl,es:[bx+2]   {cl: b23 b22 b21 b20 b19 b18 b17 b16}
   mov dh,es:[bx+1]   {dh: b15 b14 b13 b12 b11 b10  b9  b8}
   mov dl,es:[bx+0]   {dl:  b7  b6  b5  b4  b3  b2  b1  b0}
   mov ch,0          {ch:  0  0  0  0  0  0  0  0}
   test al,128       {al: b31  0  0  0  0  0  0  0}
   jz @00            {Jump if negative}

      {sign:1, al:8, cl:8, dh:8, dl:8, 00:8, ah:8}
   xor al,255    {b31 b30 b29 b28 b27 b26 b25 b24}
   xor cl,255    {b23 b22 b21 b20 b19 b18 b17 b16}
   xor dh,255    {b15 b14 b13 b12 b11 b10 b09 b08}
   xor dl,255    {b07 b06 b05 b04 b03 b02 b01 b00}
   add dl,1;   adc dh,0;    adc cl,0;    adc al,0;    mov ch,128

@00:mov ah,128+32   {sign:1, al:8, cl:8, dh:8, dl:8, 00:8, ah:8}
   test al,255;      {Check for all zeros}  jnz @3

   mov al,cl ; mov cl,dh ; mov dh,dl ; mov dl,0 ; mov ah,128+8+8+8
   test al,255; jnz @3

   mov al,cl ; mov cl,dh ; mov dh,00 ;          mov ah,128+8+8
   test al,255; jnz @3

   mov al,cl ; mov cl,00 ;                 mov ah,128+8
   test al,255; jnz @3

   mov al,00 ;                          ; mov ah,0
   jmp @57          {Finished}

  @3:                  test al,128;     jnz @57
    dec ah;            test al,64;      jnz @56
    dec ah;            test al,32;      jnz @55
    dec ah;            test al,16;      jnz @54
    dec ah;            test al,8;       jnz @53
    dec ah;            test al,4;       jnz @52
    dec ah;            test al,2;       jnz @51
    dec ah
```

```
;                {al, cl, dh, dl, 00, ah}
      rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @51: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @52: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @53: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @54: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @55: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @56: rcl dl,1; rcl dh,1; rcl cl,1; rcl al,1
   @57: and al,127; or al,ch; les bx,s;          mov es:[bx+5],al
      mov es:[bx+4],cl;    mov es:[bx+3],dh;   mov es:[bx+2],dl
      xor al,al;          mov es:[bx+1],al;   mov es:[bx+0],ah
  end; {Asm}
end; {Proc Convert}



     Procedure Ack_Lo(var CNT:byte);   begin asm    {$G+}
     mov dx,0336h; mov ah,0
   @0: dec ah; jz @1; in al,dx;   test al,128 {bit7};   jnz @0
   @1: les bx,CNT;   mov es:[bx],ah end end;


     Procedure Ack_Hi(var CNT:byte);
     begin asm   {$G+}
     mov dx,0336h;  mov ah,0
   @0: dec ah; jz @1;  in al,dx;   test al,128 {bit7};   jz @0
   @1: les bx,CNT;   mov es:[bx],ah  end end;


     Procedure Get_Result(var a1,a0:byte);
     begin asm  {$G+} mov dx,0334h; les bx,a0;  in al,dx;  mov es:[bx],al
     mov dx,0334h;  les bx,a1;  in al,dx;  mov es:[bx],al end end;



     Procedure Ext_Sign_Bit(var a0:byte);
     begin asm {$G+} les bx,a0; mov al,es:[bx];
     and al,15; test al,8; jz @0; or  al, 240
   @0: mov es:[bx],al end end;



  Procedure Int(var n2:byte; var n4,n3:byte);
  begin asm  {$G+} les bx,n2; mov al,es:[bx]; mov ah,al;
```

```
and al,112 {Mask channel};ror al,4; les bx,n4; mov es:[bx],al;mov al,ah;
and al, 12 {Mask set}; ror al,2;  les bx,n3; mov es:[bx],al end end;


Procedure C16(var count:word; var n0,n1:byte);
begin asm  {$G+} les bx,n0; mov al,es:[bx]; les bx,n1; mov ah,es:[bx]
les bx,count; mov es:[bx],ax end end;


Procedure get(var n0:byte);
begin asm {$G+} mov dx,0336h;in al,dx;les bx,n0;mov es:[bx],al end end;


Procedure GetReading (var nn:I88; var Count:Word; var n4,n5 :Byte );
var j,k,n3,n2,n1,n0,CNT :Byte;
begin {l 1}
error := false;
j:= 0;
n5:=0;          n2 := 0;
while  (j < 7) do
 begin   {j 2}
    inc(n5);

    k := 0;
    n4 := 0;
    while (k<3) do
      begin  {k 3}
       inc (n4);
       if (( n2 and 128) = 0) then
         begin   {0 4}
            setport6_7(Port6);
            ack_hi(CNT);          {Wait for ack to go high}
            if (CNT =0) then  exit;
            get_result (n2,n0);
            int(n2,j,k);
            nn[j,k+4] := n2;
            nn[j,k  ] := n0
         end     {0 4}
      else
         begin   {1 4}
            resetPort6_7(Port6);
            ack_lo(CNT);         {Wait for ack to go low}
            if (CNT =0) then exit;
```

```
            get_result (n2,n1);
            int(n2,j,k);
            nn[j,k+4] := n2;
            nn[j,k  ] := n1 ;
          end;   {1  4}

      end; {k 3}

{if n4 <> 4 then error := true;}

  end;   {j  2}
  setPort6_7(Port6);    {*}
  resetPort6_7(Port6);

  if n5 <> 8 then error := true;

  c16(count,nn[6,3],nn[7,3]);

end;



    Procedure ReadShaftEncoder_Azi (var S:seal;var P:byte; var T_Word:word);
    begin asm    {$G+} {Do not change bits P.1 and P.0}
    les bx,P     ; mov ah,es:[bx];     and ah, 0fch {ah:0000 00nn}
;
    mov dx,301h;
    mov al,ah; or al,bit2; out dx,al {al:0000 01nn} {Set Update Command}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;

    les bx,t_word;
    mov al,ah;   out dx,al {al:0000 00nn} {Reset Update Command}
;
    mov dx,331h
 @0: in al,dx     ;    and al, bit6 ;    jz @0;  {Wait for update complete}
;
    mov dx,301h                              {Set address 0}
    mov al,ah     ;    or al, bit4  ;    out dx,al {al:0001 00nn}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
```

```
    mov dx,332h  ;   in al,dx ;   mov es:[bx],al ; {Read LSByte}
;
    mov dx, 301h                    {ReSet address 0}
    mov al,ah    ;   out dx,al              {al:0000 00nn}

                              {Set address 1}
                or al,bit3  ;   out dx,al  {al:0000 10nn}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;

    mov dx,332h  ;   in al,dx ;   mov es:[bx+1], al ; {Read MSByte}

    mov dx, 301h                    {Reset Address 1}
    mov al,ah    ;   out dx,al    {al:0000 00nn}

  end; {asm}
  S := t_word ;
 end; {procedure}

    Procedure ReadShaftEncoder_Ele (var S:seal;var P:byte; var T_Word:word);
    begin asm   {$G+} {Do not change bits P.1 and P.0}
    les bx,P     ; mov ah,es:[bx];     and ah, 0fch {ah:0000 00nn}
;
    mov dx,301h;
    mov al,ah; or al,bit5; out dx,al {al:0000 01nn} {Set Update Command}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;

    les bx,t_word;
    mov al,ah;   out dx,al {al:0000 00nn} {Reset Update Command}
;
    mov dx,331h
  @0: in al,dx     ;   and al, bit7 ;   jz @0;  {Wait for update complete}
;
    mov dx,301h                          {Set address 0}
    mov al,ah    ;   or al, bit7  ;   out dx,al {al:0001 00nn}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
```

```
    mov dx,330h  ;   in al,dx ;   mov es:[bx],al ; {Read LSByte}
;
    mov dx, 301h                    {ReSet address 0}
    mov al,ah    ;   out dx,al              {al:0000 00nn}

                              {Set address 1}
                or al,bit6  ;   out dx,al  {al:0000 10nn}

    {Delay for 1 us}
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
    nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;

    mov dx,330h  ;   in al,dx ;   mov es:[bx+1], al ; {Read MSByte}

    mov dx, 301h                    {Reset Address 1}
    mov al,ah    ;   out dx,al    {al:0000 00nn}

  end; {asm}
  S := t_word ;
 end; {procedure}


  Procedure Wheel_Pulse;
  begin asm {$G+}mov dx,0306h;
          mov al,0;out dx,al;mov al,bit0;out dx,al end end;



  Procedure Wheel_Read(var Port9_In:Byte);
  begin asm {$G+} les bx,Port9_In; mov dx,0305h;
     in al,dx; mov es:[bx],al end end;


  Procedure Sensors (var wx,wz,wy,joy_o,joy_p,  c_azi, c_ele  :seal;
                  var Count:word);
  var j,k:byte;
  begin
     GetReading (    nn, Count, n4,n5    );
     j := 3; if (error=false) then begin
     for k := 0 to 7 do nnn[k] := nn[j,k]; Convert(zzzz,nnn) end;wx:=-zzzz;

     j  := 7;if (error=false) then begin
     for k := 0 to 7 do nnn[k] := nn[j,k]; Convert(zzzz,nnn) end;wz:=-zzzz;

     j := 6; if (error=false) then begin
```

```
    for k := 0 to 7 do nnn[k] := nn[j,k]; Convert(zzzz,nnn) end;wy:=-zzzz;

    j := 2; if (error=false) then begin
    for k:=0 to 7 do nnn[k]:=nn[j,k]; Convert(zzzz,nnn) end;joy_O := zzzz;



    j := 4; if (error=false) then begin
    for k:=0 to 7 do nnn[k]:=nn[j,k]; Convert(zzzz,nnn) end;c_azi := zzzz;


    j := 0; if (error=false) then begin
    for k:=0 to 7 do nnn[k]:=nn[j,k]; Convert(zzzz,nnn) end;c_ele := zzzz;



    j  := 5;if (error=false) then begin
    for k:=0 to 7 do nnn[k]:=nn[j,k]; Convert(zzzz,nnn) end;joy_P := zzzz;
end;



Procedure Draw(x,y:seal);
var a,b:integer; c:boolean;
begin
 c:=false; G_Set(ApP,BpP,C);
 A :=-Trunc(X*117)+255;    B := Trunc(Y*100)+255; G_Set(a,b,c);
 ApP := a; BpP:= b;
end;



 Procedure Fit(var ZZZ,ZZ,Z,YZ,SZ:seal);
 var Y9:seal;
 begin YZ :=(5*z+zz+zz-zzz)/6; y9 := (z+zz+zzz)/3; SZ := (YZ - Y9) end;



(* Procedure GetPort77(var a,b:byte);
  begin
  a:=0; b:=255;
  while a<>b do
  begin asm {$G+}mov dx,0331h; les bx,b; in al,dx; mov es:[bx],al;
      les bx,a; in al,dx; mov es:[bx],al end end end;
*)
```

```
  Procedure Encoders(var SaeO, SaeP:Seal);
  begin
   ReadShaftEncoder_Azi( SaeO, Port6,temp_word);  {0 <= SaeO <= 65535}
   ReadShaftEncoder_Ele( SaeP, Port6,temp_word);  {0 <= SaeP <= 65535}
   saeo:= saeo + saeo_c; saep := saep + saep_c;
   if saeO>32767 then saeo:=saeo-65535; if saep>32767 then saep:=saep-65535;
   SaeO :=  SaeO*ConversionToRadians;  SaeP := -SaeP*ConversionToRadians
  end;

(*
   Procedure GetPortD(var value,sam:byte);
  begin sam:=0;value:=255;while sam <> value do begin asm {$G+}mov dx,019dh;
  les bx,value;in al,dx; mov es:[bx],al;les bx,sam;in al,dx; mov es:[bx],al
  end end end;
*)

(*  Procedure PutPortC(nnnn:byte;var value:byte); begin value:=nnnn;
  asm {$G+} mov dx,019ch; les bx,value; mov al,es:[bx]; out dx,al end; end;
*)

(*   Procedure BitGet(var a,b:byte);
  begin asm {$G+}
  les bx,B; mov al, es:[bx]; and al,40h; add al,al; mov ah,al
  les bx,A; mov al, es:[bx];  or al,ah;  mov es:[bx],al end; end;
 *)


  Procedure SerialToReal(var s:ar128b;var v:real);
  begin
  asm {$G+} les bx,S; mov ch,es:[bx  ]; mov cl,es:[bx+1]; mov dh,es:[bx+2];
  mov dl,es:[bx+3]; mov ah,es:[bx+4];mov al,es:[bx+5];
  les bx,V; mov es:[bx  ],ch; mov es:[bx+1],cl; mov es:[bx+2],dh;
  mov es:[bx+3],dl; mov es:[bx+4],ah;mov es:[bx+5],al;end; end;



  Procedure ConvertSing(n:word; var s:ar128b; var x:real);
  var m,k:word; t:ar128b;
  begin m:=1; for k := n to n+1 do begin t[m]:=s[k+k-1];inc(m);t[m]:=s[k+k];
  inc(m); end; t[6]:=t[4];t[5]:=t[3]; t[4]:=t[2];t[3]:=0; t[2]:=0;
  SerialToReal(t,x); end;
```

```
Procedure ConvertReal(n:word; var s:ar128b;var x:real);
var m,k:word; t:ar128b;
begin m:= 1; for k := n to n+2 do
    begin t[m] := s[k+k-1]; inc(m); t[m] := s[k+k]; inc(m);
end;
serialToReal(t,x)
end;



Procedure PutPort6(var a:byte);
begin asm {$G+} mov dx,0301h;les bx,a; mov al,es:[bx]; out dx,al; end end;


Procedure Delay_(a:word);
var i:word;
begin for i := 0 to a do begin end end;


Procedure SuperElevation(var Range, Angle:Seal);
{Units are meters and radians}
begin  Angle := Range*(0.000003119+Range*0.00000001521)/2 end;
(*

Procedure LaserRangeFinder;
begin
        begin
           Port6 := Port6 and (255-32);
           PutPort6(Port6);
           Delay_(200);
           Port6 := Port6 or 32;
           PutPort6(Port6);
;
           Port7Bit6 :=0;
           while Port7Bit6 = 0 do
                 begin
                    GetPort7(Port7In);
                    Port7Bit6 := Port7In and 64;
                    if keypressed=true then Port7Bit6:=1;
                 end;
               GetPortC(PortC0);
               port6 := port6  and 255 -  8;
               PutPort6( Port6 );
               delay_(400);
               GetPortC(PortC1);
```

```
                Port6 := port6  or 8;
                dist := 0;
                if portc0 and 128 > 0 then dist:= dist + 200;
                if portc0 and  64 > 0 then dist:= dist + 2000;
                if portc0 and  32 > 0 then dist:= dist + 4000;
                if portc0 and  16 > 0 then dist:= dist + 8000;
                if portc0 and   4 > 0 then dist:= dist + 400;
                if portc0 and   2 > 0 then dist:= dist + 800;
                if portc0 and   1 > 0 then dist:= dist + 1000;

                if portc1 and  64 > 0 then dist:= dist + 20;
                if portc1 and  32 > 0 then dist:= dist + 40;
                if portc1 and  16 > 0 then dist:= dist + 80;
                if portc1 and   8 > 0 then dist:= dist + 100;
                if portc1 and   2 > 0 then dist:= dist + 5;
                if portc1 and   1 > 0 then dist:= dist + 10;
                range := dist + 0.1;
                if range < 5 then range := 5;
        end; {Reading the Laser Range Finder}
 end;
*)

 Procedure Time_Of_Flight(var a, b: seal);
 begin a:= b*(0.00092314 + b*0.00000109913) end; {b denotes the range}


 Procedure MatrixIntegrate; begin
 Mat_Der( xi, et, ze, ch,   wx, wy, wz,   xidp, etdp, zedp, chdp);
 Integrate(xi,et,ze,ch,  xid0,xidp, etd0,etdp, zed0,zedp, chd0,chdp);
 Up_Date(xid0, xidp,  etd0, etdp,  zed0, zedp,  chd0, chdp);
 Normalize(xi, et, ze, ch);  {Assures orthonormality}
 if WC then Wheel_Pulse;  {Allows time for the embedded controller}
 Mat( xi, et, ze, ch, A); {Defines the A transformation matrix} end;


 Procedure DriftCorrection; begin wx := Coeff1 * (wx/count - DriftX);
 wy:=Coeff1*(wy/count - DriftY);  wz := Coeff1 * (wz/count - DriftZ); end;


 Procedure DAC00(var xx:integer); begin (*asm {$G+}
        mov dx,0ffe0h; les bx,xx; mov ax,es:[bx]; xor ah,008h
        out dx,ax end*) end;

 Procedure DAC01(var xx:integer); begin(* asm {$G+}
        mov dx,0ffe2h; les bx,xx; mov ax,es:[bx]; xor ah,008h
```

```
        out dx,ax end*) end;

  Procedure DAC02(var xx:integer); begin (*asm {$G+}
        mov dx,0ffe4h; les bx,xx; mov ax,es:[bx]; xor ah,008h
        out dx,ax end*) end;

  Procedure DAC03(var xx:integer); begin (* asm {$G+}
        mov dx,0ffe6h; les bx,xx; mov ax,es:[bx]; xor ah,008h
        out dx,ax end*) end;

 Procedure ConfigA(var z:byte);{J3}
  begin asm{$G+}
  mov dx,0303h;{Configuration A}les bx,z;mov al,es:[bx];out dx,al end end;

  Procedure ConfigB(var z:byte);{J4}
  begin asm{$G+}
  mov dx,0307h;{Configuration B}les bx,z;mov al,es:[bx];out dx,al end end;

  Procedure ConfigC(var z:byte);{J3}
  begin asm{$G+}
  mov dx,0333h;{Configuration C}les bx,z;mov al,es:[bx];out dx,al end end;

  Procedure ConfigD(var z:byte);{J4}
  begin asm{$G+}
  mov dx,0337h;{Configuration D}les bx,z;mov al,es:[bx];out dx,al end end;


(*--------------------------------MAIN-------------------------------------*)
{   The origin for the Body Fixed Coordinates (BFCs) is the sight
aligned with the orientation of the buggy.
xx, yy and zz define the location of the sight in Inertial Coordinates (ICs).
xw, yx and zw define the location of the counter wheel in ICs.
xb, yb and zb define the location of the target wrt the BFCs.
xa, ya and za define the location of the target wrt the ICs.}

BEGIN {main}   G_Init; {Initializes the video display board}

ww:=$80;ConfigA(ww);
ww:=$82;ConfigB(ww);
ww:=$9b;ConfigC(ww);
ww:=$99;ConfigD(ww);
app:=0;bpp:=0; {Initializes the graphics}
Port6 :=127;
GetPort7(P7In);
writeln('main ');
```

```
GPS:=FALSE; WC:=true;{ if((P7In and 4)=0) then GPS := TRUE else WC := TRUE;}

  ao := 0; ap := 0; {Dummy variables used by the stepper procedure}

  JoyO_:=0; JoyP_:=0; {Used in the joystick integration mode}

  C_Joy_O:=pi/(180*131072);C_Joy_P:=pi/(180*131072);{Joy stick sensitivity}

  Flop:=1; SaeO_:= 0; SaeP_ := 0; OO:=0;  O:=0;  PP:=0;  P:=0;

  sensors(wy,wz,wx,joyp,joyo,c_azi, c_ele, Count);  {Done for synchronization}

  c_a_drift := c_azi/count;        c_e_drift :=  c_ele/count;

  get_result (n2,n0);              ii_c_azi:=0;

  Ack_Lo(CNT); {Test slave's output - low normal state}
  if (CNT = 0) then
   begin
     writeln( 'Slave in wrong state.  Press any key');
     halt;
   end;
  writeln('sensors ');

  sum1 := 0; sum2 := 0; B1 := 0; B2 := 0;
;
  ResetPort6_0(Port6);    Fir := False;  LLL := 0;

  xc:=0; dt := 0.0060; dt2 :=dt/2;
  xid0:=0; etd0:=0; zed0:=0; chd0:=0;
  Speed:=0;
  Range := 100; {Center of Screen (COS) Sight to Target distance}
  {xb, yb, zb define the target in Body Fixed Coordinates (BFCs)}
  Encoders (SaeO,SaeP); {Center of the Screen} {O elevation,  P azimuth}
  CosP := cos(SaeP); SinP := sin(SaeP);CosO := cos(SaeO); SinO := sin(SaeO);
  xb:=CosO*CosP*Range;
  yb:=CosO*SinP*Range;
  zb:=SinO*Range;{COS, wrt BFCs, BFCs}

  if WC then
   begin
     Wheel_Pulse; {Initializes wheel pulse counter}
     xx:=0;yy:=0;zz:=0; {xx yy zz:location of the origin of the BFCs, ICs}
   end;
```

```
Initialize_Q(xi,et,ze,ch);  {0,0,0,1}
Mat( xi, et, ze, ch, A); {Defines the A transformation matrix, A=[1]}

{Initial position of the Reference Wheel, ICs} {A=[1]}
xw:=xx - WheelToSight_X; yw:=yy - WheelToSight_Y; zw:=zz-WheelToSight_Z;
{xw yw zw: the Reference Wheel wrt BFCs, ICs}

SuperElevation(Range, SE); {SE defines the super elevation angle}
ZSE := SE*Range; {ZSE becomes the apparent z offset of the target}

BFCsToICs(xa,ya,za,xb,yb,zb); {Target's Position, COS, wrt BFCs, ICs}
xxx:=xa+xx; yyy:=ya+yy; zzz:=za+zz; {COS&Range, wrt ICs,ICs}
IntP:=0; IntO:=0; yo:=0; yp:=0; so:=0; sp:=0;

i_c_azi:=0; i_c_ele:=0; {Integrals of the camera rates, azi. and ele.}

writeln(' xx  yy  zz ', xx:12:2, yy:12:2, zz:12:2);
writeln(' xb  yb  zb ', xb:12:2, yb:12:2, zb:12:2);
writeln(' xa  ya  za ', xa:12:2, ya:12:2, za:12:2);
writeln('xxx yyy zzz ', xxx:12:2, yyy:12:2, zzz:12:2);
Looper := 0; {Used by the wheel counter to automatically go to drift mode}
writeln('Entering master loop');
(*
 for lll:= 1 to 40 do
 begin
  sensors(wy,wz,wx,joyp,joyo,c_ele, c_azi, Count);  {Done for synchronization}
  c_a_drift := c_azi/count;
  c_e_drift :=  c_ele/count;
  driftx:=wx/count; drifty:=wy/count; driftz:=wz/count;
  c_a_drift := c_a_drift + ( c_azi - c_a_drift)*0.001;
  c_e_drift := c_e_drift + ( c_ele - c_e_drift)*0.001;
  driftx:=driftx+(wx-driftx)*0.001;
  drifty:=drifty+(wy-drifty)*0.001;
  driftz:=driftz+(wz-driftz)*0.001;

 end;
*)
   sensors(wy,wz,wx,joyp,joyo,c_ele, c_azi, Count);  {Done for synchronization}
   driftx:= wx/count; drifty:=wy/count; driftz:=wz/count;
    c_azi:=c_azi/count; c_ele:=c_ele/count;

 driftx:= 41; drifty:= -753; driftz :=356   ;
c_a_drift:=2157; c_e_drift:=574;

vxx := 0; vyy := 0 {- 4.47} {10 mph}; vzz:=0; {Speed of the target in BFCs.}
```

```
t:=0;
   weppp:=0; wepp:=0; wep:=0;

   wappp:=0;wapp:=0; wap:=0;


clrscr;
(* - - - - - - - - - - - - - Master Loop - - - - - - - - - - - - - - - - - -*)

while true do
 begin

   if keypressed=true then
        begin writeln(driftx:16:6,drifty:16:6,driftz:16:6,
        c_a_drift:16:6,c_e_drift:16:6); halt end;

   GetPort7(P7In);

   Sensors(wy, wz, wx, JoyP, JoyO, c_ele, c_azi, Count);

   c_azi := c_azi/count;          c_ele := c_ele/count;
   c_azi := c_azi - c_a_drift;    c_ele := c_ele - c_e_drift;


   DriftCorrection;  {Cancels out static drift in wx, wy and wz}

;  {Looper is used to turn on and off the drift corection}
   {and is based on when the last wheel pulse was detected.}

   if ((Looper=0) and (P7In and 1 = 0 )) then
    begin
     if wx < 0 then driftx := driftx - 0.2 else driftx := driftx + 0.2;
     if wy < 0 then drifty := drifty - 0.2 else drifty := drifty + 0.2;
     if wz < 0 then driftz := driftz - 0.2 else driftz := driftz + 0.2;
     end;
     if c_azi<0 then c_a_drift := c_a_drift - 1.2 else c_a_drift := c_a_drift + 1.2;
     if c_ele<0 then c_e_drift := c_e_drift - 1.2 else c_e_drift := c_e_drift + 1.2;

   MatrixIntegrate; {Update Quaterions based on angular rates}
;
   if WC then
    begin
      Wheel_Read(Port9_In); {Number of pulses since last update}
      Temp := Port9_In*0.133; {0.133 is the distance in meters per count}
      BFCsToICs(xx,yy,zz,WheelToSight_X,WheelToSIght_Y,WheelToSight_Z);
```

```
      xw := xw + Temp*a[1,1];xx:= xx + xw; {xw: Ref. Wheel's position, ICs}
      yw := yw + Temp*a[1,2];yy:= yy + yw; {xx: Scope's position, ICs}
      zw := zw + Temp*a[1,3];zz:= zz + zw;
      {xw, yw and zw: the Reference Wheel's position in IC}
      {xx, yy and zz: the location of the sight (origin of BFCs) in ICs}
      if Port9_In > 0 then looper:=1815 else if looper>0 then dec(looper);
      Speed:= 0.99*Speed + 0.01*temp/dt;
    end;
;
{   gotoxy(1,1); write(speed:12:2);   }

;
   Encoders (SaeO,SaeP); {Center of the Screen} {O elevation,  P azimuth}
   CosP := cos(SaeP); SinP := sin(SaeP);CosO := cos(SaeO); SinO := sin(SaeO);
;

   {Routine for setting the Range}
   GetPortB(PortB_In);

   If (PortB_In and 4=0) then
   begin     {manual ranging and Target Position (BFCs) update}
     IntO:=C_Joy_O*JoyO/count; IntP :=-C_Joy_P*JoyP/count;
     i_c_ele:=0; i_c_azi:=0; {Terms used for the joystick integration}
     xw:=0; yw:=0; zw:=0;
     If ( (PortB_In and 8) = 0 ) then  Range := Range + 0.1;
     If ( (PortB_In and 16) = 0 ) then  Range := Range - 0.1;
     if range < 5 then range := 5;
     xb:= CosO*CosP*Range ; yb :=  CosO*SinP*Range;  zb := SinO*Range;
     BFCsToICs(xa,ya,za,xb,yb,zb);{Center of Screen (COS) & Range, wrt BFCs}
     xxx:=xa+xx;yyy:=ya+yy;zzz:=za+zz;{COS&Range, wrt ICs,ICs}
     t:=0;

   end;
{gotoxy(1,1);
writeln(xxx:12:2,yyy:12:2,zzz:12:2);
writeln( xx:12:2, yy:12:2, zz:12:2);
 }


;
   {Target Acquire}

   if((PortB_In and 4>0) and ((PortB_In and 16)=0)) then
    begin
     { LaserRangeFinder;}
```

```
      if range <10 then range:=100;
      IntO:= C_Joy_O*JoyO/count; IntP := -C_Joy_P*JoyP/count;
      i_c_ele:=0; i_c_azi:=0; {Terms used for the joystick integration}
      xw:=0; yw:=0; zw:=0;
      xb := CosO*CosP*Range; yb :=  CosO*SinP*Range; zb := SinO*Range;
      BFCsToICs(xa,ya,za,xb,yb,zb);
      xxx:=xa+xx; yyy:=ya+yy; zzz:=za+zz;{COS&Range, wrt ICs,ICs}
      sensors(we0,we1,we2,we3,we4,we5,we6, we7);

      t:=0;
    end;
    ;
{    t:=t + dt;}

{    xxx := xxx + vxx*dt; yyy := yyy+vyy*dt ; zzz := zzz+ vzz*dt;}

{gotoxy(1,1);   writeln(t:12:2,xxx:12:2, yyy:12:2,zzz:12:2); }

   RRR:=XXX - XX; SSS:=YYY - YY; TTT:=ZZZ - ZZ; {COS&Range, wrt BFCs, ICs}
   Range:=sqrt(sqr(RRR) + sqr(SSS) + sqr(TTT)); {COS distance}

{    Output(4,Range);   }
;
    Time_Of_Flight(TimeOfFlight,Range);
    {TimeOfFlight is an approximation neglecting the speed of the vehicle}
;
    ICsToBFCs(rr,ss,tt,rrr,sss,ttt);{COS&Range+Gravity, wrt BFCs (BFCs&ICs)}
    azi0 := atan(SS,RR); ele0 :=atan(TT,sqrt(sqr(RR)+sqr(SS))); {COS}
;
    SuperElevation(Range, SE); {SE defines the super elevation angle}
    ZSE := SE*Range; {ZSE becomes the apparent z offset of the target}
;
 {   dxx:=vxx*TimeOfFlight ; dyy:= vyy*TimeOfFlight; dzz:=vzz*TimeOfFlight;}

  {  ICsToBFCs(dx,dy,dz,dxx,dyy,dzz);}


    RR:=RR + Speed*TimeOfFlight {- DX};   {RR SS TT: hit point, wrt BFCs, BFCs}
  {  SS := SS - DY;    }
    TT:=TT - ZSE {- DZ} ; {Gravity Drop of the Projectile}
    {COS&Range+Gravity, Lead Angle, wrt BFCs}

;
    azi1:=atan(SS,RR){+0.238/range}; ele1 :=atan(TT,sqrt(sqr(RR)+sqr(SS)));
;
```

```
    SetPort6_0(Port6);

    {2O} Stepper_Driver(0,AO,YO+0.5*SO,SO);

    Slew := 0.005; if ((PortB_In and 64)=0) then slew := 0.05;
    JoyP := C_Joy_P*JoyP/count;
    if ((P7in and 2)=0) then IntP := IntP + JoyP*slew;
    JoyP_ := JoyP + IntP;
;
    Slew := 0.005; if ((PortB_In and 8)=0) then slew := 0.01;
    JoyO := - C_Joy_O*JoyO/count;
    if ((P7in and 2)=0) then IntO:=IntO + JoyO * slew {else t:=0};
    JoyO_ := JoyO + IntO;

    {2P} Stepper_Driver(1,AP,YP+0.5*SP,SP);
;
    SaeO := SaeO + JoyO_; {Elevation}  Saep := Saep + JoyP_; {Azimuth}
    D_El0 :=(SaeO - Ele0);          D_Az0 := (SaeP - Azi0);
    D_El1 :=(SaeO - Ele1);          D_Az1 := (SaeP - Azi1);
    {Elevation, O,  Azimuth, P}

    OOO := OO; OO:=O; O :=-D_El0;     PPP := PP; PP:=P; P := D_Az0;
    YO9:=YO; Fit(OOO,OO,O,YO,SO) ;  {Generates YO,SO}
    YP9 := YP; Fit(PPP,PP,P,YP,SP);  {Generates YP,SP}

    {0O}  Stepper_Driver(0,AO,YO,SO);

   ele_lim:=0.0030;    azi_lim:=0.0030;    tau:=0.4;


   con1:=dt;  con2:=1-con1;

   c_azi:=c_azi*0.000010;
   c_ele:=c_ele*0.000010;


   i_c_ele := (i_c_ele + c_ele *con1)*con2;
   i_c_azi := (i_c_azi + c_azi *con1)*con2;


{gotoxy(1,1); write(c_azi:12:6, i_c_azi:12:6,c_a_drift:12:6);}

   if i_c_ele >  ele_lim then i_c_ele :=  ele_lim;
   if i_c_ele < -ele_lim then i_c_ele := -ele_lim;
   if i_c_azi >  azi_lim then i_c_azi :=  azi_lim;
```

```
    if i_c_azi < -azi_lim then i_c_azi := -azi_lim;

    azza:= -   i_c_azi;
    azze:=     i_c_ele;

 {write('      azza1 ',azza:8:4);   }
    weppp:=wepp; wepp:=wep; wep:= -D_El1;
    wappp:=wapp;wapp:=wap; wap:=  D_Az1;


    draw(       weppp + azze ,
              wappp + azza );
;

    for m := 1 to 5 do r[m-1]:=r[m];
    r[5]:=o;
    john := r[5]+(r[5]-r[0]);

    for m := 1 to 5 do v[m-1]:=v[m];
    v[5]:=p;
    jane := v[5]+(v[5]-v[0]);
;
    FirP := Fir;
    If ( (P7In and 32=0) and (Fir=False) ) then
    begin   err:=sqrt(sqr(john)+sqr(Jane));  if err<0.0002 then Fir:=True  end;

    if ((Fir = True) and (FirP = False)) then  SetPort6_1(Port6);
    if  (Fir = True) then inc(LLL);
    if  (LLL =  10) then ResetPort6_1(Port6);
    if  (LLL =  100) then begin Fir:=False; ResetPort6_1(Port6);LLL:= 0; end;

;
    if fir then john := john+ 0.002;
;
    if fir then jane := jane + 0.002;


    if (o > ANG) then xc:=$7ff else if (o < -0.05/3) then xc:=$800 else
    xc := trunc(  temp * 40960*3);  DAC00(xc);



{ *** }  temp2:= - weppp{- c_azi*0.002};
    if (p > ANG) then xc:=$7ff else if (p < -0.05/3) then xc:=$800 else
    xc := trunc(   temp2 *40960*3 );  DAC01(xc);
```

```
    if (john > ANG) then xc:=$7ff else if (john < -0.05/3) then xc:=$800 else
    xc := trunc(john*40960*3);  DAC02(xc);

    if (jane > ANG) then xc:=$7ff else if (jane < -0.05/3) then xc:=$800 else
    xc := trunc(jane*40960*3);  DAC03(xc);
;

    GetPort7(P7In);
    P7InP := P7In;

  (*  while ( not (P7In and 8 = 8) and (P7InP and 8 = 0) ) do
    begin P7InP:=P7In;GetPort7(P7In) end; {External Clock Synchronization}
 *)
    ResetPort6_0(Port6);
    {0P}  Stepper_Driver(1, AP,YP,SP);
   end;
end. {main}
```

INTENTIONALLY LEFT BLANK.

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC DDA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDQ D SCHMIDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	OSD OUSD(A&T)/ODDDR&E(R) R J TREW THE PENTAGON WASHINGTON DC 20301-7100
1	DPTY CG FOR RDA US ARMY MATERIEL CMD AMCRDA 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN PO BOX 202797 AUSTIN TX 78720-2797
1	DARPA B KASPAR 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1	NAVAL SURFACE WARFARE CTR CODE B07 J PENNELLA 17320 DAHLGREN RD BLDG 1470 RM 1101 DAHLGREN VA 22448-5100
1	US MILITARY ACADEMY MATH SCI CTR OF EXCELLENCE DEPT OF MATHEMATICAL SCI MADN MATH THAYER HALL WEST POINT NY 10996-1786

NO. OF COPIES	ORGANIZATION
1	DIRECTOR US ARMY RESEARCH LAB AMSRL DD 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CS AS (RECORDS MGMT) 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1145
	ABERDEEN PROVING GROUND
4	DIR USARL AMSRL CI LP (BLDG 305)

NO. OF COPIES ORGANIZATION

3 CDR
USASOC
DSCR
AOFI RI MSI
E BROWN SOST COORDINATOR
FORT BRAGG NC 28307

2 CDR
USASOCOM
SOST T
W WILLIAMS
BLDG 102
MACDILL AFB FL 33621-5316

1 OFFICE OF SPECIAL TECH
G SHOCK
10530 RIVERVIEW RD
FT WASHINGTON MD 20744

5 CDR
US ARMY ARDEC
AMSTA CCJ
S SMALL
PICATINNY ARSENAL NJ
07806-5000

2 CDR
US ARMY ARDEC
AMSTA DSA SA
J UNTERKOFLER
M DOWNES
PICATINNY ARSENAL NJ
07806-5000

ABERDEEN PROVING GROUND

5 DIR USATC
STEAC AE CA
P MCCALL
D GRIFFIN
STEAC FC M
A ROSE
G NIEWENHOUS
G BREWER

NO. OF COPIES ORGANIZATION

ABERDEEN PROVING GROUND (CONT)

7 DIR USARL
AMSRL WM BA
W D'AMICO
T BROSSEAU
B HAUG
M KREGEL
J MCLAUGHLIN
AMSRL WM MB
R KASTE
L BURTON

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2000	Final,

4. TITLE AND SUBTITLE
The Inertial Reticle Technology (IRT) Applied to an M16A2 Rifle Firing From a Fast Attack Vehicle

5. FUNDING NUMBERS
1L162618AH80

6. AUTHOR(S)
Timothy L. Brosseau, Mark D. Kregel, Baily T. Haug, and John T. McLaughlin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-WM-BA
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2209

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Motion of the muzzle of a weapon fired from a moving vehicle occurs during firing because of many factors, such as vibrations caused by the vehicle's wheels or the terrain. This motion can have adverse effects on the capabilities of the weapon to hit a target because the shooter is unable to accurately position the muzzle of the weapon onto the target as the projectile exits the barrel. Large, heavy vehicles, such as the Abrams tank, the Bradley Fighting Vehicle, and the costly Apache helicopter, have very expensive gun turrets that are controlled by very expensive, fully stabilized gun sights to accurately position the muzzle of the weapon onto the target. However, small and lightweight vehicles, such as a small helicopter, a fast attack vehicle, or a high-mobility multipurpose wheeled vehicle (HMMWV), cannot justify such expensive gun turrets and fully stabilized sights. Therefore, to improve the accuracy of a weapon firing from a small, lightweight vehicle, the U.S. Army Research Laboratory (ARL) has developed the Inertial Reticle Technology (IRT).

This report presents how the IRT was applied to a 5.56-mm M16A2 rifle firing from a fast attack vehicle. The complete details of the IRT applied to a 5.56-mm M16A2 rifle firing from a fast attack vehicle are presented along with an analysis of stationary and moving vehicle live fire test data.

14. SUBJECT TERMS
Inertial Reticle Technology, M16A2 rifle, fast attack vehicle, M855 ammunition, video camera, flat display monitor, quartz rate sensors, shaft angle encoders, firing solenoid

15. NUMBER OF PAGES
56

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-2209 (Brosseau) Date of Report April 2000

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name E-mail Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)
(DO NOT STAPLE)